Transactions

/*Transactions

are the mechanism used to group a set of SQL statements together such that either all or none of the statements succeed.*/

/*Multiuser Databases

These DBMSs support two or more two users accessing the database simultaneously. Multi-user systems contain all the mini-computers and mainframe computers. In a mainframe computer, the database may exist on a single computer, and in other computers, the database may be distributed on multiple computers. Multiple users can update data while working together simultaneously.

Example: Databases of Banks, insurance agencies, stock exchanges, supermarkets, etc. */

/*Locking

Locks are the mechanism the database server uses to control simultaneous use of data resources. When some portion of the database is locked, any other users wishing to modify (or possibly read) that data must wait until the lock has been released.

Lock Granularities

There are also a number of different strategies that you may employ when deciding how to lock a resource.

Table locks: Keep multiple users from modifying data in the same table simultaneously

Page locks: Keep multiple users from modifying data on the same page (a page is a segment of memory generally in the range of 2 KB to 16 KB) of a table simultaneously

Row locks: Keep multiple users from modifying the same row in a table simultaneously*/

/*transaction, which is a device for grouping together multiple SQL statements such that either all or none of the statements succeed (a property known as atomicity). To protect against this kind of error, the program that handles your transfer request would first begin a transaction, then issue the SQL statements needed to move the money from your savings to your checking account, and, if everything succeeds, end the transaction by issuing the commit command. If something unexpected happens, however, the program would issue a rollback command, which instructs the server to undo all changes made since the transaction began.*/

/***Starting a Transaction**

Unless you explicitly begin a transaction, individual SQL statements are automatically committed independently of one another. To begin a transaction, you must first issue a command.*/

/*Ending a Transaction

Once a transaction has begun, whether explicitly via the start transaction command or implicitly by the database server, you must explicitly end your transaction for your changes to become permanent. You do this by way of the commit command, which instructs the server to mark the changes as permanent and release any resources (i.e., page or row locks) used during the transaction.*/

/*Transaction Save points

In some cases, you may encounter an issue within a transaction that requires a rollback, but you may not want to undo all of the work that has transpired. For these situations, you can establish one or more save points within a transaction and use them to roll back to a particular location within your transaction rather than rolling all the way back to the start of the transaction.*/

```
SHOW TABLE STATUS LIKE 'customer';
```

/*InnoDB

A transactional engine employing row-level locking For any tables that might take part in transactions, however, you should choose the InnoDB engine, which uses row-level locking and versioning to provide the highest level of concurrency across the different storage engines.*/

```
ALTER TABLE customer ENGINE = INNODB;
```

/*Exercsie - 1

Generate a unit of work to transfer $50 from account 123 to account 789. You will need to insert two rows into the transaction table and update two rows in the account table. Use the following table definitions/data:

Account:

account_id	avail_balance	last_activity_date
123	500	2019-07-10 20:53:27
789	75	2019-06-22 15:18:35

Transaction:

txn_id	txn_date	account_id	txn_type_cd	amount
1001	2019-05-15	123	C	500
1002	2019-06-01	789	C	75

Use txn_type_cd = 'C' to indicate a credit (addition), and use txn_type_cd = 'D' to indicate a debit (subtraction).*/

```
/*START TRANSACTION;
INSERT INTO transaction
(txn_id, txn_date, account_id, txn_type_cd, amount)
VALUES
(1003, now(), 123, 'D', 50);
INSERT INTO transaction
(txn_id, txn_date, account_id, txn_type_cd, amount)
VALUES
(1004, now(), 789, 'C', 50);
UPDATE account
SET
    avail_balance = available_balance - 50,
    last_activity_date = NOW()
WHERE
    account_id = 123;
UPDATE account
SET
    avail_balance = available_balance + 50,
    last_activity_date = NOW()
WHERE
    account_id = 789;
COMMIT;*/
```